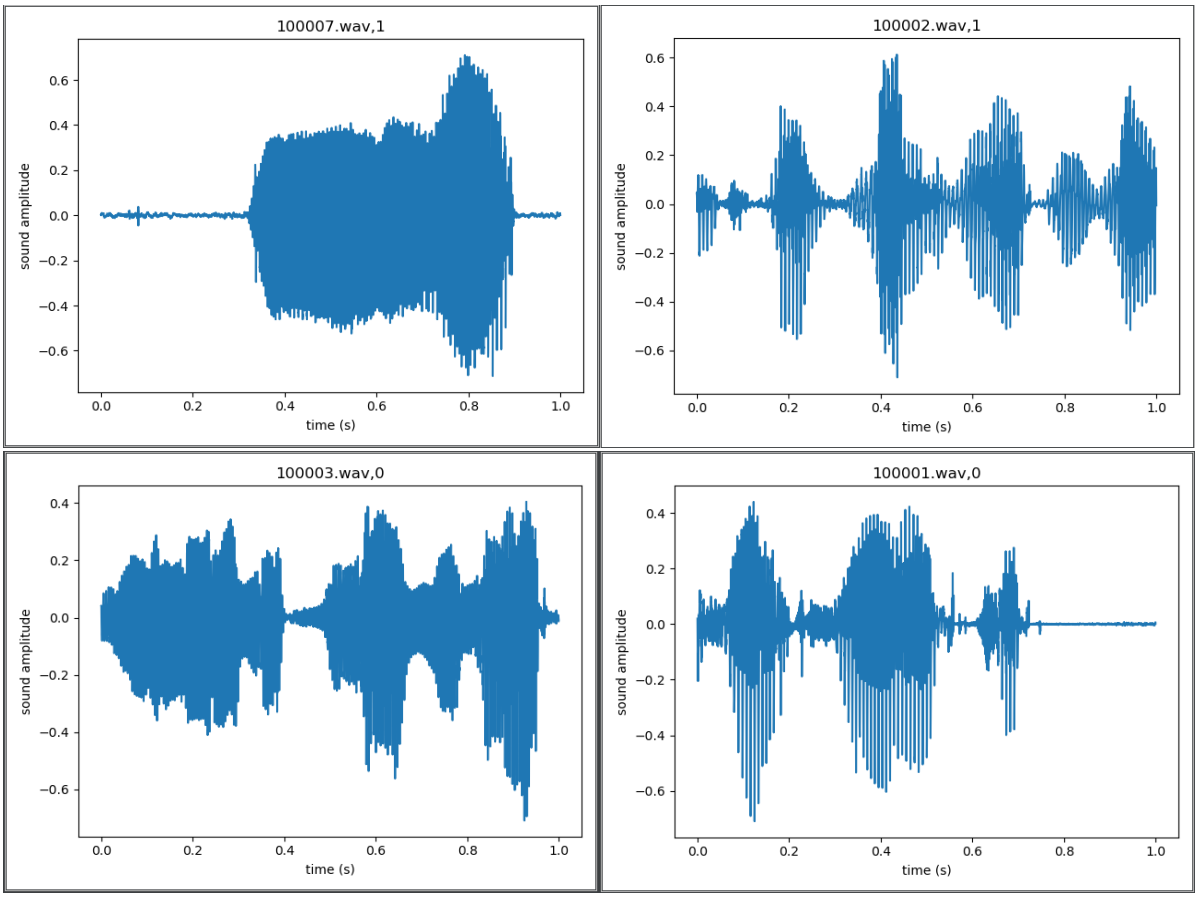
Surgical mask detection

Student: Pantiru Luana – Cătălina

Profesor îndrumător: Găman Mihaela

Acest proiect se bazează pe clasificarea unor fișiere audio în care sunt persone care vorbesc cu sau fără mască chirurgicala, folosind algorimti de machine learning.

Primul pas pe care l-am facut a fost să vizualizez datele date. Am primit 8000 de fișiere audio pentru train, 1000 pentru validation și 3000 pentru test. Toate aceste fișiere aveau o durată de o secundă. Fișierele audio care sunt clasificate că persoana care vorbește are mască chirurgicală au label-ul 1, iar cele fără mască au label-ul 0. Am plotat cate două fișiere pentru fiecare label.



Pentru a putea citi fișierele audio în python am folosit din biblioteca librosa funcția load(), iar ca să îmi extrag feature-urile, funcția mfcc(). Funcția mfcc() are

rolul de a transforma fierace probă audio într-o imagine. Dupa aplicarea funcției mfcc am facut o medie pe fiecare linie a matricii care a reieșit pentru a avea datele mai compacte[1]. Atât feature-urile, cât și label-urile sunt memorate fiecare într-o listă pentru fiecare set de date: train, validare, test.

Primul algoritm pe care l-am folosit pe setul de date a fost Naïve Bayes, folosind ca și clasificator Bernoulli. Am ales acest clasificator în urma mai multor teste cu diferiti clasificatori, cum ar fi Gaussian sau Multinomial. Folosind acest algoritm nu aveam o acuratețe mai mare de 0,53. Acest algoritm este rapid și ușor de folosit, dar are pretenția ca predictorii să fie independenți ceea ce nu este valabil pe setul nostru de date.

Următorul algoritm pe care l-am folosit a fost SVM(Support Vector Machine). Ca să îmi performez algoritmul, m-am folosit de hiperparametrii: C este parametrul de regularizare, puterea de normalizare este invers proporțională cu C; kernel are rolul de a lua datele de intrare și de a le transforma în forma dorită; gamma reprezintă coeficientul lui kernel. Ca să aleg parametrii potriviți am tot testat diferite valori pentru C, kernel și gamma. Cautând diferite informații despre SVM, am găsit că îmi pot tuna parametrii folosind GridSearchCV[2]. GridSearchCV este o funcție implementata în sklearn.model_selection care caută pe baza modelului combinația cea mai bună a valorilor hiperparametrilor. Este nevoie de un model, în cazul nostru SVM, și un dicționar în care sunt date diferite valori pentru hiperparametrii. Astfel, această funcție va face toate combinațiile posibile, ca la final să ne afișeze valorile care să ne ajute să opținem o acuratețe mai bună. Pe datele mele mi s-a generat urmatoarea combinație : C = 10, kernel = 'rbf', gamma = 1, iar datele au fost normalizate folosind MinMaxScaler din sklearn.preprocessing, având pe datele de validare o acuratețe de 0.77. Cu toate acestea, varianta finală pe care am folosit-o pe kaggle a fost: C=10, kernel='linear', gamma=, datele fiind normalizate cu StandardScaler. Folosind aceste valori, pe datele de validare aveam o acuratețe de 0.696, iar pe leaderboardul final 0.63761. Am ales aceaste valori deoarece pe leaderboardul public aveam rezultate mai bune decat pe cel cu acuratețea de 0.77 pe leaderboardul public.

Cea mai vizibilă diferență dintre cei doi algoritmi este rezultatul acurateții pe datele de validare. Această diferență este cauzată de modul în care sunt folosite

[1] https://medium.com/@mikesmales/sound-classification-using-deep-learning-8bc2aa1990b7
[2] https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/

datele de train. După cum am specificat, tratează datetele independent, în timp ce SVM-ul caută interacțiunile dintre date.

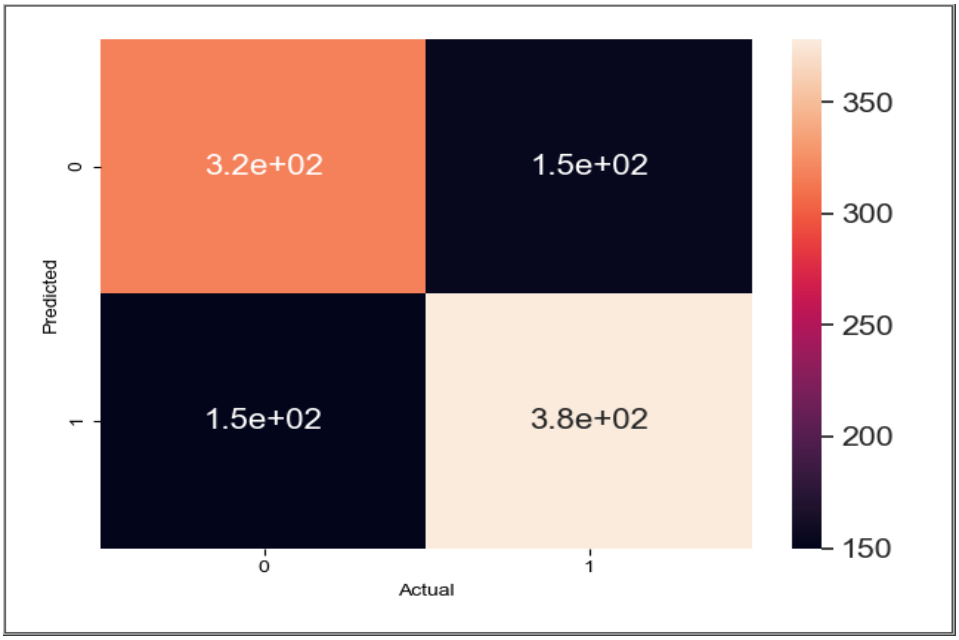
Pentru că eram curioasă cum se modeleaza datele și pe alți algoritmi, am încercat să folosesc și MLPClassifier. Comparativ cu ceilalți algoritmi pe care i-am prezentat deja, acest algoritm se bazează pe o rețea neurală pentru a îndeplini sarcina de clasificare. Și acesta la randul lui are hiperparametrii cu care poți jongla pentru ați îmbunătăți algoritmul: alpha care reprezintă paramtrul de regularizare; learning_rate_init este rata de învățare; max_iter reprezintă numarul maxim de epoci pentru antrenare; solver este regula de învatare; hidden_layer_size reprezintă numărul de neuroni din al i-lea strat ascuns; activation este funția de activare pentru stratul ascuns; momentum este actualizarea descendenței gradientului și este folosit când solver = ‘sgd’. Folosind aceeași tehnică ca la SVM de a găsi o combinație destul de bună pentru hiperparametrii, adică GridSearchCV. După apelul acestei funcții mi s-au generat urmatoarele valori: activation= 'relu', alpha= 0.005, hidden_layer_sizes= (100, 100), learning_rate_init= 0.01. Ca acuratețe pe datele de validare s-a identificat o creștere față de SVM, având pe acest algoritm 0.79.

De-a lungul competiției am avut diferențe destul de mari între acuratețea pe datele de validare și scorul dat pe kaggle în urma predicțiilor de pe datele de test. Aceasta diferență a fost în jur de 0.20, pe MLPClassifier. M-am gandit ca poate gresesc eu ceva la cod, mai precis la prelucratul datelor. Luând asta în considerare am modificat numarul de mffc-uri care să fie returnate de la 40 la 128. Am ajuns la numarul de 128 prin încercarea valorii 200 și mi-a fost dată o eroare legată ca nu se poate introduce o valoare de dimensiune 128 într-o variabilă cu shape 200. Astfel am micșorat diferanța la 0.17.

```
  File "C:/Users/Admin/Desktop/tema/tema.py", line 112, in <module>
    audio_validation, label_validation = formare(validare, array1, label_validation)
  File "C:/Users/Admin/Desktop/tema/tema.py", line 62, in formare
    aux[i] = mfccsscaled
ValueError: could not broadcast input array from shape (128) into shape (200)
```

Cu toate acestea tot nu s-a dovedit un algoritm ales bun pentru datele mele, având pe scorul public 0.62333 și pe cel privat 0.61714. Astfel am rămas ca model final folosit SVM-ul.

Matricea de confuzie pentru modelul final este:



Și f1 score: 0.7132075471698113

Acestea au fost calculate cu funcțiile specifice, confusion_matrix și f1_score, din biblioteca sklearn.metrics.